Node.js提速指南

没有人奢望基于Node.js的各类系统能够一统web服务器这一庞大领域，但Node自身所具备的灵活性确实使其身影屡屡出现在各种各样的任务处理流程之中。那么到底是哪些特色使得Node从以往那些web框架与平台中脱颖而出呢?

作者：核子可乐译 来源：51CTO | 2011-11-10 08:55 收藏 分享

【51CTO经典译文】Node.js又被简称为Node，作为一款针对web开发者推出的web应用程序平台，它已经在过去的一年中得到了相当令人满意的普及度。没有人奢望基于Node.js的各类系统能够一统web服务器这一庞大领域，但Node自身所具备的灵活性确实使其身影屡屡出现在各种各样的任务处理流程之中。那么到底是哪些特色使得Node从以往那些web框架与平台中脱颖而出呢?归纳起来有两点，基于事件以及JavaScript。

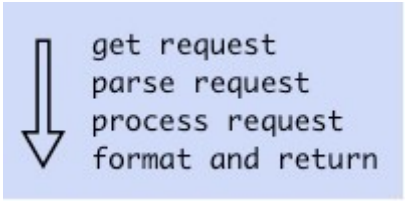
51CTO推荐专题：Node.js专区

编辑推荐

热点 惹毛程序员的十件事！需求变更居然不是排第一！

头条 高性能Java持久化的14个技巧

基于事件



传统的阻断程序

传统的web请求处理流程分为几步，即接收请求、进行解析、等待必要资源载入、处理(无论需要多长时间)以及返回响应。由于整个过程中充斥着大量等待环节，因此在同时处理两个或者两个以上请求时必须为每个请求分配一个独立的线程来满足执行需求。需要处理的请求越多，我们需要的线程就越多，同时我们还要为每个线程的管理投入大量额外的资源。

基于事件的框架则采取完全不同的解决方案，不过这类框架同时也要求我们使用不同的编码方式。它们所利用的正是许多服务器应用程序浪费在等待I/O上的时间，尝试将这些时间用在实实在在的工作上。执行线程实际上只有一个，但程序员将自己的代码有机地加以分解，并把每一块视为一次“事件”的出现。举例来说，打开一个文件当然会带来I/O时间，因此在以事件为主导的系统中，我们会下达“请开始打开一个文件，当文件打开工作完成后，再调回该功能。”这样框架就会着手打开文件，并把过程中需要用到的功能记录下来，最后等待操作系统发来的文件打开完成通知。一旦通知送达，该事件将立即被触发，转而调用所需功能。

```
open_file(filename, open_done_function)

open_done_function() { write(open_file, data, write_done_function) }

close_file_function() { close(open_file) }
```

某种风格的非阻断事件驱动虚拟代码

现在大家的第一反应可能是：“这难道不会使我的代码在互连功能方面变得乱七八糟吗?”答案是肯定的，如果大家的代码在表现力上有所不足，那么这种负面现象的确会发生。举例来说，如果各位选择的是能够处理匿名函数的语言，那么代码可能会变成如下所示：

```
open_file(filename, function(file) {
     write(file, data, function(file) {
          close(file) }
     ) }
);
```

另一种风格的非阻断事件驱动虚拟代码

真正的区别在于，执行时间在操作完成时并不会立即中止，而是会在底层操作系统通知任务结束后记录接下来所要进行的任务。事件驱动类编程能够通过一系列语言实现，可读性与易用性也能够加以控制——就Node.js而言，最佳候选语言无疑是JavaScript。

JavaScript

Node.js是为使用V8 JavaScript引擎所量身定制的，这款引擎同时也是Chrome浏览器的核心。该引擎被用于为Node提供执行环境，并完成准时化编译及其它优化项目。JavaScript语言从历史层面来说名声并不算好，因为它被过多地滥用于浏览器动画处理，并且与名称中的Java毫不相干。不过在过去的十年中，开发人员们已经开始发现JavaScript所具备的强大能力，尤其是在Scheme、Self以及Lisp方面，比起诸如Java、C与Pascal等程序语言，JavaScript的表现可以用惊艳来形容。将它视为Java的变种只能说是种历史的误区。这种重新审视的起点来自由Douglas Crockford所撰写的《JavaScript：好的一面》，该文以Crockford的亲身工作体会及同名讲座为基础(详见以下谷歌技术讲座视频)。

JavaScript: 好的一面, 是由Doug Crockford推出的一次谷歌技术讲座

Crockford指出，JavaScript故意提高了程序员对其的熟知感，但这也同时导致不少人误以为不需要进行有针对性的学习;在这种论调的基础上，他们大多没有学习必要的基本理念。自以为了解JavaScript，或者说自认有能力用它为网页添加编程功能的家伙不在少数。但他们最后往往会惊讶地发现，大多数JavaScript编码，例如JavaScript中的每一个对象，实际上是一套关联数组。尽管JavaScript运行缓慢的恶名已经广为人知，但在近期的浏览器大战中大部分竞争者都在努力为所有的浏览器添加JavaScript加速机制。这个过程的意义在于像V8这样的JavaScript引擎的诞生，同时也让更多编程人员真正开始熟悉JavaScript。将对JavaScript语言本身的理解及经过加速的JavaScript引擎进行结合，这种高效的结合体必然能为我们带来更加光明的应用前景……

创造历程

Ryan Dahl, Node.js的创造者，其灵感来源于在利用Ruby网页服务器以上传文件为目的进行进度更新时所遇到的各种不便与问题。“这么简单的事情执行起来居然如此复杂，这让我非常惊讶，” Dahl在2010年的一次采访中如是说。**而动态Ruby网页服务Mongrel则给他留下了深刻的印象。**这款由JavaScript“军备竞赛”所提供的语言让Dahl相信，web开发人员能够在它的帮助下更便捷地在浏览器中进行工作;而他也将自己在事件驱动服务方面的知识与该语言结合起来，创建出了Node的最初版本。由于JavaScript缺乏服务器端库，Dahl与其它开发人员一道创造出了服务器端的JavaScript标准库。这就使得Node.js具备了凌驾于其它各类事件驱动型框架之上的优势，因为其库在编写中始终贯穿着事件驱动这一理念;而在其它框架中，我们可以轻易发现那些标准库无法调用的非事件痕迹，这种情况往往会导致创建受阻。

生态系统

Node.js的飞跃使得诸多在业界内处于领先地位的web开发者们将其作为自己的原型系统后端。这种普及的顺利实现得益于可重用库的功能性生态系统。NPM，即Node工具包管理器，目前其中已经罗列了超过四千四百种工具包，其中较为常用的有像Underscore这样的通用库、像request这样的简化库以及像Jade这样的模板引擎。在以一套库支持大部分Node.js应用的队伍中，最引人注目的当数Socket.io：它利用适当连接自动选取技术，实现了客户端与服务器之间的实时连接;根据浏览器的不同功能，它所采用的机制也分WebSockets、AJAX查询、AJAX流等。这套库使动态web应用程序的创建更为简便，它去除了辨别浏览器及管理连接工作中所固有的复杂性，并因此广泛受到希望规范新的动态web应用程序的开发者们的青睐。

Node.js应用程序更有趣的一点是其作为应用平台的出现;在WebOS 2.1中，该操作系统的开发者们将其添加到自己的移动平台之上，借以创建利用JavaScript为本地应用程序编写的本地服务项目。另一项值得称道的应用就是其运行于最新发布的BeagleBone中，用户可以登录其上，借助网络创建Node.js脚本，进而控制该平台及I/O端口。

Node.js获得的来自生态系统的最新助力源于Node.js 0.6.0，这是首个具备Windows端口的Node版本。这很可能进一步扩大Node的潜在发展空间。

优点与缺点

人们常会说Node.js具备相当良好的可扩展性，其实对于Node.js而言，其中还有更深层次的特定含义：Node.jf，无论是内部还是其本身，比起很多其它技术都能够更好地在单核心处理器上高效应对I/O绑定任务。但是，当打算使用更多的核心、或者希望通过更多系统运行Node.js应用程序时，我们必须回到较为传统的扩展性处理模式：将Node.js应用程序运行于多个核心或系统中，并在其上设置负载平衡项目，以将工作量有效分散。因此，尽管将

Node.js扩展到网络上确实可行，但开发人员们最终还是要将全部底层终端技术部署到同一套网页服务基础设施中来。也就是说，要让Node.js高效利用每个核心，也要为之部署很多不必要的分布式基础设施。

Node.js所无法替代的是web扩展应用程序中那些用于执行繁重任务的计算及查询服务，就连V8这样的高效JIT JavaScript编译器也无法胜任此类工作。同样重要的是，我们必须认识到Node.js并不是一款能够作用于任何应用程序的全新通用型平台，它只是现代系统架构中的一种重要解决手段;这一手段最重要也是最有效的作用是作为“web粘合剂”存在，充当其它各类web应用程序的连接组件。Node.js允许开发人员迅速创建这种连接效果，一方面是因为大家对这种语言比较熟悉，另一方面则是事件驱动模式使大家能够更加高效地利用有限的资源。综上所述，Node.js应该作为对语言及平台开发人员常用工具的有益补充。当然根据同样的思路，开发人员也完全可以出于个人的喜好而用Node.js完成整个应用程序的编写;这款框架有力地挑战了传统观念中“JavaScript无法完成某些任务”的固有思维。

接下来，为了满足乐于进一步钻研的读者朋友，我们将对Node.js服务进行一次走马观花式的浏览，并提供一些特定的Node.js资源。

简单的Node.js 服务实例

作为Node.js代码实例，这里我们列举一个略微过度设计的“世界你好”程序，该程序通过读取/tmp文件夹中的文件找出招呼的来源：

```
1. var http = require('http');
2.  var fs = require('fs');
3.  http.createServer(function(req, res) {
4.      fs.readFile("/tmp/hellomsg.txt", function(error, text) {
5.          res.writeHead(200, {'Content-Type': 'text/plain'});
```

```
            if (error) {
                res.end('Not ready to say hello...\n');
            }
            else
        {
                res.end('Hello '+text+'\n');
            }
    });
}).listen(1337, "127.0.0.1");
console.log('Server running at http://127.0.0.1:1337/');
```

逐行检测将使我们从这里观察到更多信息。

```
var http = require('http');
 var fs = require('fs');
```

JavaScript不具备任何可以作用于模块的工具包系统，因此Node.js使用的是Common.js协议以完成库操作。Require函数将检索库并返回一个句柄;var http=将该值保存在“http”当中。接下来我们重复“fs”处理，也就是文件系统库。这些句柄可以用于在库内部调用函数，如下列代码所示...

```
http.createServer(function (req, res) {
```

这里还有很多其它内容。http库被要求创建一套HTTP服务器。当该服务器上存在任何类型的请求时，所调用的都是定义过的函数。请注意，所有对象都能够被传递给函数：此处请求对象将包含请求生成的信息，而响应对象则会被由请求所产生的程序响应所填充。

但首先我们需要获取打招呼目标的文本信息...

```
1. fs.readFile("/tmp/hellomsg.txt", function(error, text) {
```

fs.readFile 函数需要一个能够读取的文件名，这样它在读取文件时就能够调用我们即将指定的回调函数。该回调函数会传递两个值，其内容不是错误信息就是文件内容。在我们继续下一步之前，需要简化整个流程并开始创建响应。响应对象同样具备协助建立响应的函数：

```
1. res.writeHead(200, {'Content-Type': 'text/plain'});
```

在这里，我们写入响应头。正在添加的是HTTP状态200，之后媒体描述内容也将以纯文本的形式写入。现在要做的工作是写入响应内容并将响应发回服务器以继续传递...

```
1. if (error) {
2.         res.end('Not ready to say hello...\n');
3.     }
4.     else
5.  {
6.         res.end('Hello '+text+'\n');
7.     }
```

res.end 函数是来自http库的实用速记位;它可以写成

```
1. res.write('Hello '+text+'\n');
2.  res.end();
```

res.end出于便于响应的目的而进行了扩展，其中一个字符串中通过允许该函数通过最终语句的形式包含了该响应。Res.end函数同时还向服务器发送了一个信号，指明该响应已经生成完毕，随时可以发送。

```
});
 }).listen(1337, "127.0.0.1");
```

根据函数的定义，创建得出的http服务器函数列表现在可以进行调用，以通知其监听位置...

```
console.log('Server running at http://127.0.0.1:1337/');
```

而且控制台可以发送消息，显示我们正在运行中的内容。到此，我们就搭建起了一套简单的HTTP服务器。它处于连接等待状态，一旦连接形成，它就会切换为工作状态并调用我们的函数。我们的函数要求读取文件，而非中止请求;在文件打开、读取以及关闭步骤完成后，它会调用一个函数。所调用的这个函数负责以可阅读的文本生成“你好”响应并将其打包发回服务器。

Node.js 资源

◆ Nodejs.org: Node.js平台的主站点。在这里可以为Node.js下载资源代码并查看每个版本的升级文档。

◆ npmjs.org: Node软件包管理器首页，这里有NPM的详细安装说明。大家还可以在这里搜索或者浏览NPM软件包。

◆ howtonode.org: 一个专门阐释Node.js相关技术的博客。

◆ Node: Up and Running: 这是O’ Reilly出版社专门为Node.js打造的开发专题网站，大家可以在这里在线阅读大量资料。

◆ NodeGuide.com: 由Felix Geisendörfer打造的Node.js选择指南。

◆ NodeCloud.org: 一个汇总了上述及更多与Node.js相关的站点的资源目录。

原文链接：

http://www.h-online.com/open/features/The-H-Speed-Guide-to-Node-js-1363974.html

【51CTO.com独家特稿，非经授权谢绝转载！合作媒体转载请注明原文出处！】

【编辑推荐】

1. Node.js初体验
2. 淘宝袁锋：Node.js会令后端人员产生危机感
3. 如何安装Node.js
4. Node.js入门之神秘的服务器端JavaScript
5. 什么是Node.js？

【责任编辑：陈贻新 TEL：（010）68476606】